MIX/360
USER'S GUIDE

BY

D. E. KNUTH
R. L. SITES

STAN-CS-71-197
March, 1971

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

MIX/360

USER'S GUIDE

by

D. E. Knuth

R. L. Sites

MIX/360 is an assembler and simulator for the hypothetical MIX machine, which is described for example in Knuth's The Art of Computer Programming, Section **1.3.1.** The system contains several debugging aids to help program construction and verification.

The one-pass assembler accepts programs written in the MIXAL language as described in The Art of Computer Programming, Section 1.3.2. The END card terminating a MIXAL program may be followed by several control cards including a GO or BTN card which invokes the simulator. The simulator optionally prints out a trace of the program and/or a post-mortem dump of the MIX memory with counts of how often each instruction was executed. There is a limited capability for batching multiple assembly/simulations during one invocation of the system.

Deck Setup - General Case

```
//jjjjjjjj JOB (aaaakkk,bbb,0.2,1),'nnnnnn'
/* SERVICE CLASS=Q
//   EXEC MIX
//SYSIN DD *
      [    ASM    ssslll  ]
          (MIXAL Program)
   $  {    TRAC address
      [    NDMP
$     {     GO  n          (or BTN n)
      [    data
/*
```

where jjjjjjjj is your job name, aaaa is your account number, kkk is your keyword, bbb is your bin number, and nnnnnn is your name. The symbol $ denotes zero or more occurrences, and [...] denotes an optional item. Control cards (ASM, TRAC, NDMP, GO, and BTN) are explained below; these codes and MIXAL operation codes start in column 12 and the associated "address" fields start in column **17.** To batch several programs in one job, follow the last data or GO card by an ASM card and another MIXAL program, etc.

Simple Setup

```
job card
/* SERVICE CLASS=Q
//   EXEC MIX
//SYSIN DD *
      (MIXAL program)
           GO  2
          [data]
/*
```

MIX/360 Control Cards

1. ASM α . The following cards (up to the next END card) must be a MIXAL program. This deck will be assembled and the program will be loaded into an initially zero MIX memory. Afterwards MIX's registers are set to zero and it is ready to begin execution at the address specified on the END card. (Execution will begin when the next GO or BTN card is encountered.) An ASM card is automatically inserted at the beginning of the input if one is not already present. The address α is normally blank; however, the address on the first ASM card may be a six-digit number ssslll , specifying unusually large time and/or line limits. In that case, the program is given sss seconds to run, and 100 times lll lines may be printed before the

whistle is blown. The default limits are 5 seconds and 500 lines (equivalent to ASM 005005) . The limits specified on the JOB card should be greater than or equal to the ASM card limits, if you want clean termination. The job card shown above under "Deck Setup" has limits of 0.2 minutes (12 seconds) and 1 thousand lines.

2. GO α. Here α is a W-value, usually a constant, denoting a nonnegative integer n less than 4000 (α blank denotes n = 0) . A GO card starts the simulated MIX computer. It starts either at the first instruction after a previous HLT instruction, or, if this is the first GO after an ASM, at the address specified on the END card. Each instruction is traced the first n times its location is encountered; tracing is explained under How to read the listings. It is suggested that GO 2 be used until your program appears to be working. A program continues to run until (a) a HLT instruction is encountered, or (b) too many execution errors occur, or (c) the actual running time or amount of printing exceeds the set Units. (See error messages below.) In case (a), another GO card may be used to continue the program. The GO card should be followed by all the data cards which will be read by the simulated MIX program up until the time its execution stops.

3. BTN a. This is like a GO a, but the simulation starts as if MIX's "Go-button" had been pushed. (See Exercise 1.3.1-26 in The Art of Computer Programming.) The card following BTN is read into locations 0-15 in MIX code; rJ is set to zero; and execution begins with the instruction at location zero.

4. TRAC α. Here α is a W-value which specifies a location to be traced. During program execution all Store or Move operations which change this location are traced, and any instruction executed from this location is traced. (Note that an IN instruction which affects this location will not cause tracing.)

5. NDMP. The post-mortem dump will be suppressed after the next GO or BTN.

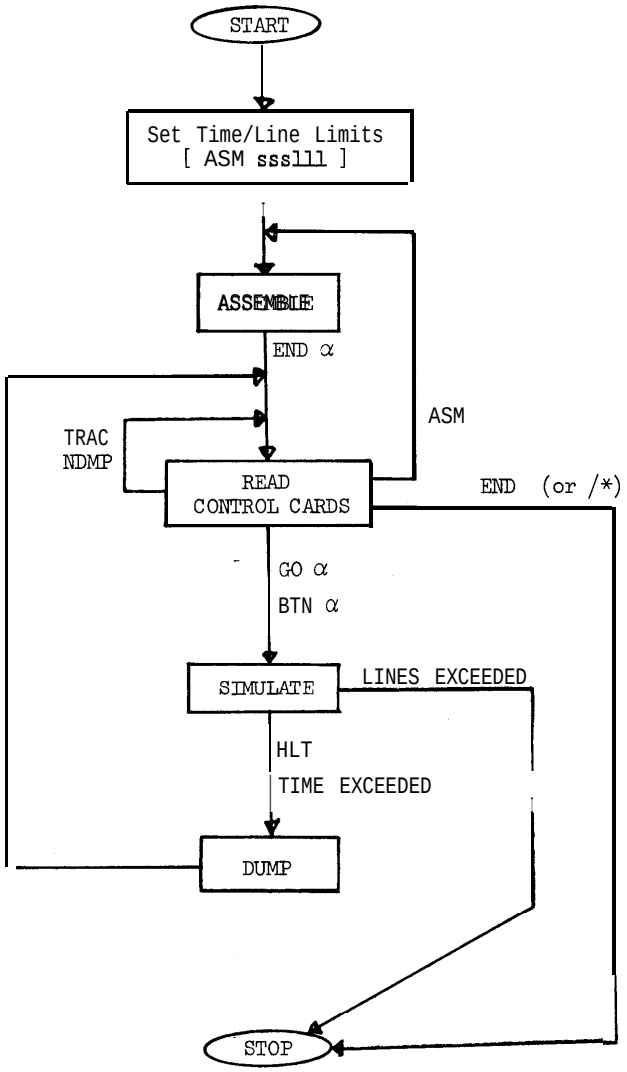
6. END a. An END card encountered during an assembly terminates the assembly. α is a W-value which specifies the location of the first instruction to be executed when a GO is encountered. If an END card is encountered- as a control card (not during assembly) the job is terminated; in this case, α is ignored. An infinite number of END cards is automatically inserted at the end of a deck immediately preceding the /* card (there is no "end-of-file" indicator), so a terminating END is generally not needed.

Batched Input

Several sets of
ASM
(MIX deck)

may be run in a single job. The time and line limits-for the whole batch is set by the first ASM card (and the job card). There are no safeguards to prevent a MIX program from reading in the next one as data. Nor are there any provisions for preventing an infinite loop in one program from terminating the whole batch before later programs are run.

Summary of the way MIX/360 looks at your deck.



Special characteristics of this MIX configuration

1. At present only unit 16 (the card reader) and unit 18 (the printer) are simulated.
2. The byte size is 100; all displayed information is in decimal notation.
3. Floating-point operations are not simulated.
4. The alphabetic character code is extended as follows:

Code	Character
10	11-0 punch (prints as a blank)
20	| (vertical line)
21	_ (underline)
56	" (double quote)
57	% (percent)
58	& (ampersand)
59	# (hash mark)
60	¢ (cents signs, prints as blank)
61	! (exclamation point, prints as blank)
62	¬ (not sign)
63	? (question mark)

How to read the listings

Suppose the (MIX deck) is:

```
*    EXAMPLE MIX DECK
READER     EQU 18
PRINTER    EQU 16
           ORIG 1000
START      IN  BUFFER(READER)
           JBUS *(READER)
           LDA  BUFFER
           LD1  =23=
1H         STA  BUFFER,1
           DEC1 1
           J1P  1B
           STZ  BUFFER+2(1:2)
           OUT  BUFFER(PRINTER)
           HLT  *
BUFFER     ORIG *+24
           END START
           Go  2
THIS IS A DATA CARD
/*
```

The output (three pages) is shown in Figures 1, 2 and 3. Notice the following points of interest corresponding to the numbers in the illustration:

1. This column shows assembled instructions or equivalents, broken into appropriate fields.
2. This column shows the current "location counter", except on EQU lines when it is blank.
3. The input is reproduced here. The vertical bars over columns 0, 10, and 16 help verify that the right card columns were punched.
4. The time and line limits are printed here (in this case 5 seconds and 500 lines).
5. A "/" just to the left of an assembled instruction denotes a "future reference" that will be fixed up later when the value of the symbol becomes known. The first occurrence of a symbol is assembled as -1 the second as the address of the first, the third as the address of the second, etc.
6. Here the "1008" means that the last "future reference" to BUFFER was in location 1008. (The loading routine will follow the links 1008, 1007, 1004, 1002, 1000, -1, changing all the addresses in these locations to the true address, 1010, of BUFFER.)
7. An error indication! It is illegal to use BUFFER+2" when BUFFER is a future reference, so the "+2" was ignored. (The programmer should either have defined BUFFER near the beginning of the program, or should use another symbol such as BUFFER2 which is later EQU'd to BUFFER+2.)
8. The total number of errors detected (1 in this case) appears here. (An attempt will be made to run the program, regardless of the number of errors found.)
9. The literal constant =23= is inserted just before the END card, in this way (note that the second = does not appear
10. This 1000 denotes the starting address computed by the END card (corresponding to START in this case).
11. The equivalent of a control card address (in this case 2) appears here. If the card "TRAC 1B" had appeared, the corresponding address would have been 1004.
12. The location of the instruction being traced.
13. The number of times this location has been encountered so far.
14. The instruction being traced.
15. Its operation code translated into symbolic form.
16. Either the address after indexing, or the contents of the word at that address (before the operation has been performed).
17. Contents of MIX's registers, before executing this instruction. (When error messages appear, the register contents after execution may be shown instead.)
18. If the overflow toggle is on, an X appears here.
19. The comparison indicator (L = less, E = equal, G = greater) .

20. The simulated time in MIX units.
21. When an instruction is encountered for the third and subsequent times, tracing is suppressed (because we said "GO 2") and a single line of dots appears. Also, in a series of consecutive NOPs only the first one is traced. All-zero words in memory are NOPs.
22. This line was output by the MIX program, not the tracing routine.
23. The final register contents upon program termination.
24. The total run time, including any time needed to terminate the last I/O operation.
25. The final contents of MIX's memory may be helpful for "post mortem" examination.
26. Here you can see the number of times each instruction was executed. For example, the instruction in location 1001 was performed **7168** times!

Assembler Error Messages

One letter error codes are printed on the left side of the assembly listing. Up to four codes will be printed for a single line.

CODE	MEANING	EXAMPLE
A	Address has wrong syntax	`LDA (1,2)`
B	Backward local symbol has not been defined	`JMP 3B`
C	Character is invalid	`JMP $`
D	Duplicate definition of location field symbol; the current definition is ignored.	`X ENTA 3` `X ENTA 4`
E	End card has non-blank location field. The symbol is ignored.	`LOC END START`
F	Field specification is improper.	`LDA XYZ(3:2)`
L	Length of symbol, constant or literal exceeds 10 characters (including first = sign in a literal).	`CON 12345678901`
M	Missing operand.	`LDA 5+(1:2)`
0	Operation code is unknown. Treated as NOP.	`J7NZ LABEL`
R	Range of location counter is wrong (< 0 or > **4000**) , or attempt to assemble a word into location 4000.	`ORIG 4001`
T	Too big a field or index specification	`MOVE XYZ( 327)`
U	Undefined symbol (future reference) used other than standing alone as an address.	`LDA XYZ(J)` where J is not defined
V	Overflow occurred during the evaluation of an expression.	`ENTA 5123456789+ 5123456789`
X	Extra operand	`LDA 2(1:1)3` the **"3"** is extra
9	The loading routine ran into trouble trying to fix up previous references to this location, either because of previous errors or an attempt to load two instructions into the same location. If this error occurs, the address fields of other instruction(s) may be incorrect also. = sign missing at the end of a literal. What do you mean? A control card was expected (ASM, END, TRAC, NDMP, GO, BTN).	
/	Not an error message, see note **5** above.	

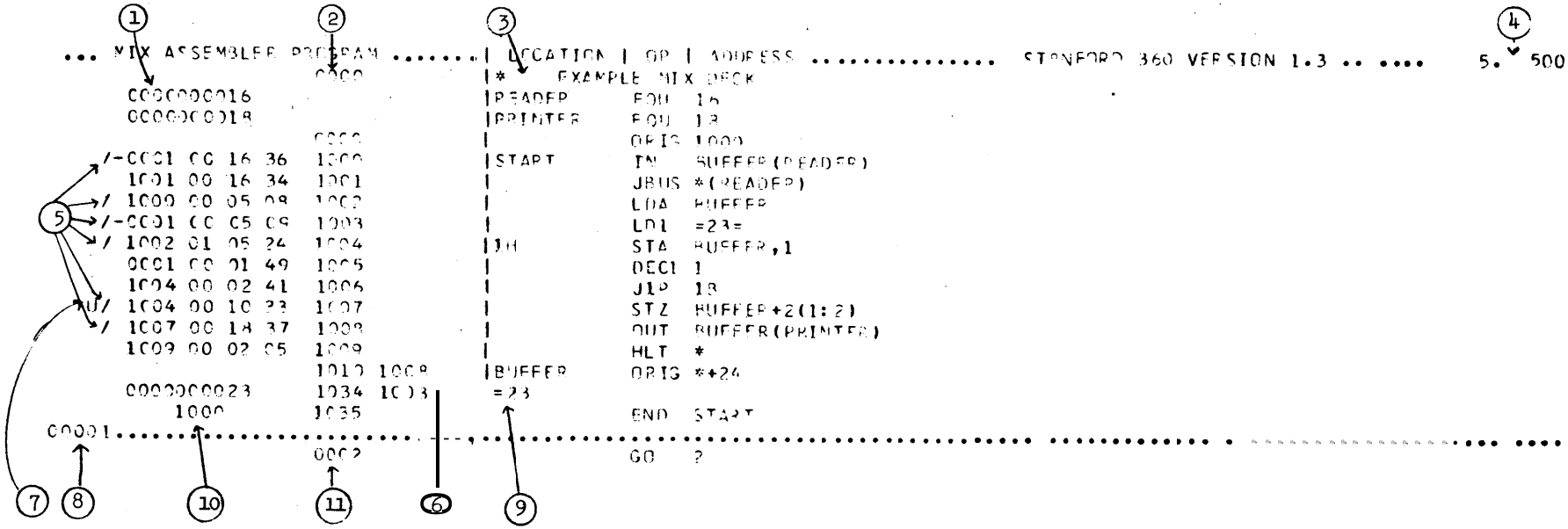


Fig. 1. First page of output: The assembled program.

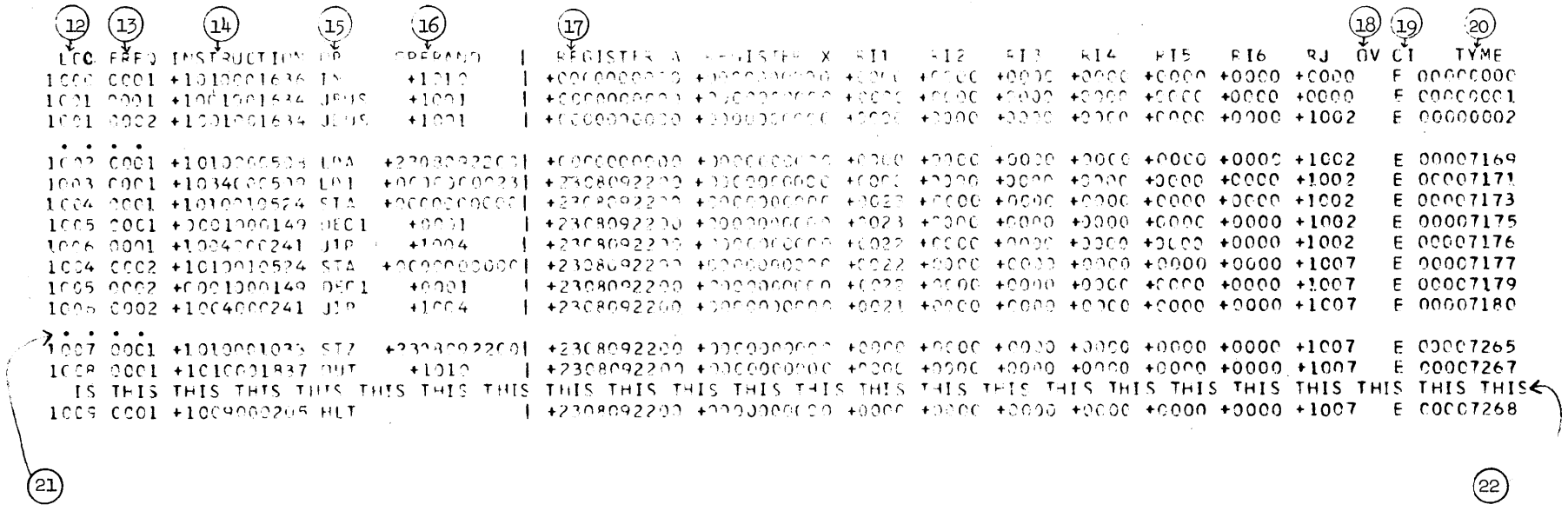


Fig. 2. Second page of output: Tracing

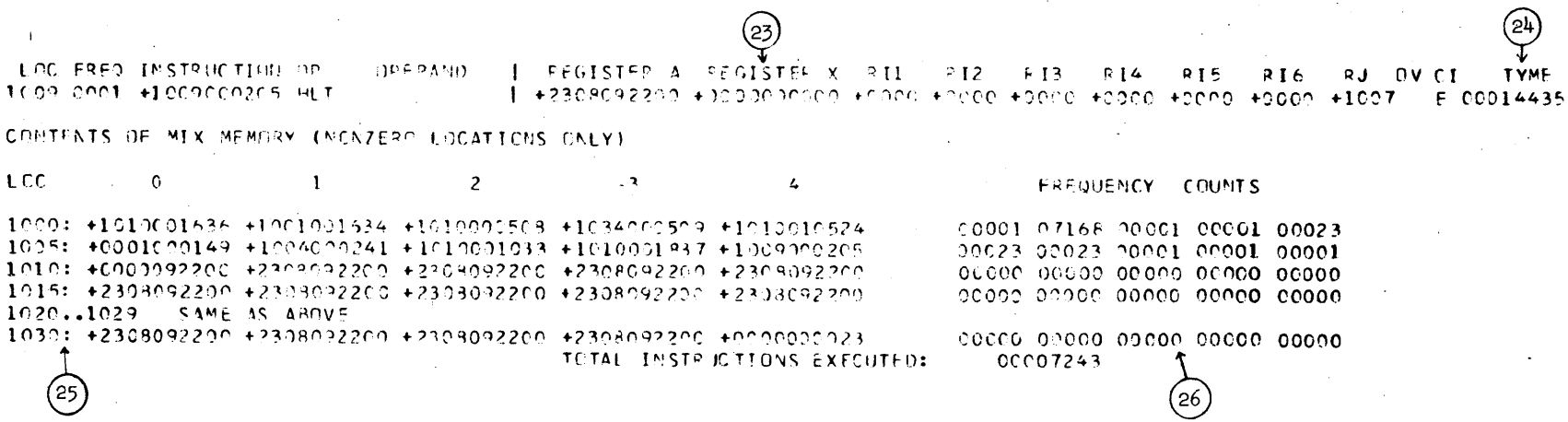


Fig. 3. Third page of output: The "post mortem".

Simulator error messages

F-FIELD IS WRONG.
The instruction has too large an F field.

I/O OPERATION INCOMPLETE.
You are trying to store into a memory area where input or output is in progress; or trying to fetch from a memory area where input is still in progress.

INCORRECT I/O UNIT.
The input/output unit you requested is not present on this version of MIX; or it is not valid for this operation (e.g. output on the card reader).

INDEX REGISTER OVERFLOW
This instruction made the index register contents more than two bytes.

INDEXING ERROR.
The I field of this instruction is greater than **6,** or the result of indexing doesn't fit into two bytes.

INVALID MEMORY ADDRESS.
The memory address is out of range (either negative or greater than 3999).

INVALID OPERATION.
The operation code is greater than **63;** or, a negative shift has been requested.

INVALID PARTIAL FIELD.
The field specification does not have the form 8L+R where $0 \leq L \leq R \leq 5$.

OPERATIONNOT IMPLEMENTED.
The present version of MIX does not include this operation (e.g. a floating point operation).

INSTRUCTION LOC IS OUT OF RANGE.
The next instruction location is negative, greater than 3999, or within an area where input is in progress.

TOO MANY HANGUPS, FUSE IS BLOWN.
Simulation stops because of excessive errors.

Other messages

TIME ISRUNNING OUT.
The 360 operating system is about to throw your program off the machine, due to lack of time, so it is necessary to terminate. (In order to increase the time allotment you must change your JOB card and use a special ASM card at the beginning.of your deck, as described above; but first make sure your program isn't in an infinite loop.)

TOO MANY LINES PRINTED.
The 360 operating system is about to throw your program off the machine, due to excessive output, so it is necessary to terminate. (If you want to increase your line allotment, you must change your JOB card and use a special ASM card at the beginning of your deck, as described above; but first make sure you are actually going to find all that output useful.)

Sane common pitfalls and how to avoid them

1. Reading between the lines of the assembly listing:

a) Undefined symbols are legal and are assigned one word each at the end of the program. Look immediately in front of the END card for these every time you assemble. These can occur from a large variety of bugs, such as using 2H instead of 2F or 2B.

b) Columns 10 and **16** are completely ignored, so address fields starting in column **16** are not flagged; the part in columns **17** on is blindly used. Likewise labels starting in column two are silently ignored. Cures -- scan your eye down the listing from the vertical bars in the heading; look for undefined symbols in (la).

c) Don't assume that the error count (1 on the sample listing) is zero. Look at it.

d) Make sure that you have the right address on the END card. If you leave it off, the simulator will execute your program starting at location zero.

e) Look for "?" on the left side of your listing. It indicates a bad control card.

f) If you get a "9" error flag, your program will not be loaded into MIX memory correctly. Execution may well be meaningless.

2. Reading between the lines of the trace output.

a) Look at the location of the first instruction traced. Did your program start at the intended address'?

b) Remember that the registers are normally those before the instruction was executed. For the result of the instruction, look at the next line. If tracing is off and an error occurs, the line before the error message will contain the registers after the instruction was executed.

c) If your program does not reach (or does not have) a HLT instruction, it will likely "fall off the end" of your program into the zeroed memory from there to location 3999. These zeros are legal NOP instructions and are traced as a single NOP followed by NEXT INSTRUCTION LOC IS OUT OF BOUNDS (trying to execute an instruction from location 4000).

d) If a particular instruction does not do what's intended, look carefully at the assembled instruction on the assembly listing and at the instruction as traced and at the post-mortem dump, to see if it was assembled as you intended (and stayed that way). Pay particular attention to the address and F fields. Remember that a partial-field Compare instruction does not work the same way as the other partial-field instructions; it uses the same field both in the register and in memory.

e) To convince yourself' that your program is working correctly, always read the trace output and rethink what is supposed to be happening. Start out by assuming that your program is incorrect, instead of assuming that it is all right.

3. Your program terminates, but you don't know where.

a) At the top of the dump page, is the LOC printed? If not, the location counter was not in the range 0 - 3999 when the program stopped. The J register contains the address of the last branch executed plus one. Say rJ is 1007; look at the trace or dump and see where the branch at location 1006 went. Verify in the dump that ,the branch has not been accidentally modified. If this branch went to a legal location, then you executed from there to location 3999.

b) Because of lines or time limit, you got no dump. Look at the last line of the trace. After it was printed, the following things may have happened: one more instruction was traced, but the printed line for it is still in a simulator output buffer, not yet on the printer; tracing was suspended because ' every instruction was executed n times (for GO n) ; the program then entered an infinite loop. This is a very common sequence (especially if you forgot to set your job card limits as high as the ASM limits), so don't use the last line of tracing as evidence that no more instructions were executed.

c) Infinite input: the MIX/360 system inserts an infinite number of END cards ("END" in columns 12-14) at the end of a deck. If there is an IN instruction inside an infinite loop, the program will eventually exceed its time or line limits. To see what is happening, always write your programs to print out their input data, e.g.

```
IN   BUFFER(16)
JBUS *(16)
OUT BUFFER(18)
JBUS *(18)
```

Implementation Notes

The MIX/360 simulator is written in 360 assembly language and takes about 80-100 microseconds on the 360/67 to interpret a single MIX instruction (with tracing off). This gives an effective speed of about 10,000 MIX instructions per second.

Each MIX memory location is kept as 10 decimal digits (5 bytes), plus sign, flags, and frequency count (3 more bytes). Decimal hardware of the 360 is used extensively.

The frequency count for each location is 20 bits, for a range of approximately 0 - 1,000,000. Simulations running over 100 seconds may overflow this count (also see JBUS below). On the post-mortem dump the counts are printed modulo 100,000, but are added into the total correctly.

I/O overlap is simulated by (1) doing the actual I/O "instantaneously" at the time the IN or OUT is interpreted, (2) setting flag bits in the memory locations involved, marking them as "I/O in progress", (3) maintaining an ordered priority queue of what TYME the next I/O operation will have its simulated completion, and (4) resetting the flag bits when the proper simulated time occurs. A simulated card read or printer write takes about 7100 tyme units; a page eject, 30,000 tyme units.

An untraced JBUS * is simulated by artificially setting the TYME to the completion tyme of the next I/O event, and incrementing the frequency count of the JBUS appropriately. Because of this special arrangement, a typical JBUS instruction will have its frequency count incremented by about 7000 in the same time that it takes the simulator to interpret one other instruction. So about 145 executions of a JBUS * loop will overflow its 2^{20} frequency counter. An overflowed counter is reset modulo 2^{20}.